Motivating High Performance Serverless Workloads

Hai Duc Nguyen
ndhai@cs.uchicago.edu
University of Chicago
Chicago, Illinois, U.S.A.

Zhifei Yang
zhifei@cs.uchicago.edu
University of Chicago
Chicago, Illinois, U.S.A.

Andrew A. Chien
achien@cs.uchicago.edu
University of Chicago and Argonne National Laboratory
Chicago, Illinois, U.S.A.

ABSTRACT

The historical motivation for serverless comes from internet-of-things, smartphone client server, and the objective of simplifying programming (no provisioning) and scale-down (pay-for-use). These applications are generally low-performance best-effort. However, the serverless model enables flexible software architectures suitable for a wide range of applications that demand high-performance and guaranteed performance. We have studied three such applications - scientific data streaming, virtual/augmented reality, and document annotation. We describe how each can be cast in a serverless software architecture and how the application performance requirements translate into high performance requirements (invocation rate, low and predictable latency) for the underlying serverless system implementation. These applications can require invocations rates as high as millions per second (40 MHz) and latency deadlines below a microsecond (300 ns), and furthermore require performance predictability. All of these capabilities are far in excess of today's commercial serverless offerings and represent interesting research challenges.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**.

KEYWORDS

Serverless; High Performance Computing; Document Annotation; Virtual Reality; Stream Processing

ACM Reference Format:
Hai Duc Nguyen, Zhifei Yang, and Andrew A. Chien. 2021. Motivating High Performance Serverless Workloads. In *Proceedings of the 1st Workshop on High Performance Serverless Computing (HiPS '21), June 25, 2021, Virtual Event, Sweden.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3452413.3464786

1 INTRODUCTION

Recent years have witnessed a rapid growth in the popularity of serverless (aka function-as-a-service or FaaS), which enables users to create functions without provisioning VM's or containers, and provides a pay-for-use model. With these advantages, serverless lowers the bar for building applications in effort, use, and cost. With the cloud provider responsible for resource management, serverless allows applications to scale rapidly to hundreds of invocations within a few seconds [19, 42]. This has been a great boon to IoT and smartphone applications. Most commercial serverless implementations focus on these smartphone clients and IoT applications. These applications are low rate per client (but can scale up for large numbers of clients) and access the serverless functions over wide area networks. They are not demanding in terms of invocation rate (per client) and invocation latency. As a result, serverless systems today require 10-250 milliseconds of software overhead per invocation [12, 42], limiting the scope of application use in both invocation rate (cost) and guaranteed performance.

Recently, serverless has been the focus of interest for a wide range of distributed applications, such as video encoding, distributed compilation, and data analytics [3, 14, 15, 19, 20, 27]. These types of intra-cloud serverless applications are driving increased performance requirements for serverless systems. These applications and more are the focus of our work.

Despite the current commercial incarnations of serverless that provide only best-effort service and incur significant overhead for invocations (limiting invocation rates), we believe that the serverless model and serverless application architectures can be applied for a wide range of demanding distributed applications. Specifically, we consider demands of high performance and strict performance guarantees. To support these applications, serverless systems will need to achieve dramatically lower overhead per invocation (high performance) and provide applications with the ability to engineer strict performance guarantees (real-time). To make this case, we have collected a set of demanding applications of the serverless model, and present them here.

We consider three applications: streaming scientific data, augmented reality, and text document annotation. For each, we describe the application and its critical performance requirements. Next, we map the application onto a serverless software architecture, describe how these requirements manifest in the serverless application architecture, and the derived performance requirements that result such as invocation rate, latency, etc. These serve as motivation and requirements for high-performance serverless systems. Next, to enable research on high-performance serverless implementations, we describe a set of parametric workload models, which characterize each application, and can be used to study both the performance of serverless implementations and their impact on application performance.

Specific contributions of the paper include

- Identification and documentation of three compelling high-performance application classes for serverless systems

HiPS '21, June 25, 2021, Virtual Event, Sweden
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8388-2/21/06…$15.00
https://doi.org/10.1145/3452413.3464786

- Exposition of a serverless architecture that illustrates how the quantitative application performance requirements and guarantees translate into quantitative performance requirements in invocation rates (millions/second) and latency guarantees (< 1 microsecond) for the serverless system
- For each we present a characterization and workload model enabling future applications and serverless systems research

The rest of the paper is organized as follows. Sections 2, 4, and 3 each describe a demanding application, its performance requirements, and a serverless architecture for that application. We close each section with a workload model that can be used for serverless system experiments. Section 5 briefly describes related work, and Section 6 summarizes the paper.

2 SCIENTIFIC DATA STREAMING

High data rate instruments – small (e.g. DNA sequencing or CAT scanners) and large (the advanced photon source or CERN's particle accelerator) produce high bandwidth streams of scientific data. These streams are increasingly reliant on real-time processing capabilities and machine learning workflows for filtering, analysis, and adaptive experiment control. Beyond high data rates, they also require low latency response for real-time control. In Particular, we focus on a challenging example from high-energy physics (HEP) to highlight how extreme these performance requirements can be.

Consider online filtering for the Large Hadron Collider (LHC) [4, 13] where the system targets at finding HEP events that may contain evidence of new subatomic physical phenomena such as dark matter. The objective is to rapidly identify the potentially interesting collisions because the rest must be discarded due to data storage limitations. Because collisions are extremely short, sampling must be taken at extremely high frequency, and filtering must in turn operate at extremely high frequency and low latency. Detailed description is provided below.

2.1 Application Description

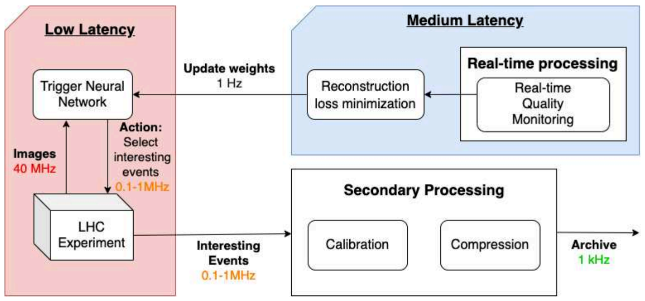


Figure 1: HEP Event Filtering and Analysis System Workflow

At the Large Hadron Collider (LHC), a stream of event records is generated by sampling internal physics processes. The LHC experiment instruments pre-process each record into an image summary. The image summaries are output at a fixed frequency of 40 MHz. A Trigger Neural Network (TNN) then classifies each image summary to tell whether it contains an interesting signal. Once it detects an interesting signal, the TNN tells LHC Experiment buffering devices to send the corresponding buffered event record into a secondary processing workflow, which may include heavy and hardware-accelerated computations, such as energy measurement, data selection, data formatting, serialization, and data archiving. Due to limited LHC buffering capacity, the TNN has to finish every detection within a hard deadline of 150-300 nanoseconds (upgrades to the LHC detectors are expected to increase this latency budget as much as 10-fold [10]. Asynchronously, there is also a real-time quality monitoring process that samples event records and periodically pushes updates to the TNN model to adjust filtering quality.

Figure 1 shows the workflow. Each part is annotated with invocation rate and latency requirements, which will be elaborated later in Section 2.4.

2.2 Examples

Besides HEP, there is a range of scientific data streaming examples with similar real-time requirements, including:

- Experiments on synchrotron facilities such as ANL's Advanced Photon Source and LBL's Advanced Light Source
- High Energy Physics, Large Hadron Collider and ATLAS experiments
- High speed genome sequencing machines
- High Resolution MRI (magnetic resonance imaging) or CAT Systems (Computed Axial Tomography)

Despite objective and workflow differences, the scientific and medical instrument that used in these examples have three key properties in common: operate at extremely high event rates, process a large amount of data, and has strict, short deadline requirement.

2.3 Mapping into Serverless Architecture

Data stream processing in scientific experiments can be naturally mapped into a event-driven serverless software architecture, and doing so improves the portability, scalability and flexibility of these workflows. As a concrete example, Figure 2 shows the serverless architecture for the LHC event filtering workload. Each double-border box represents a serverless function. The *LHC Experiment* box at the bottom refers to the event record collecting and buffering devices inside LHC, with the assumption that it can invoke serverless functions with image summaries, sampled event records, or interesting event records selected by the TNN.

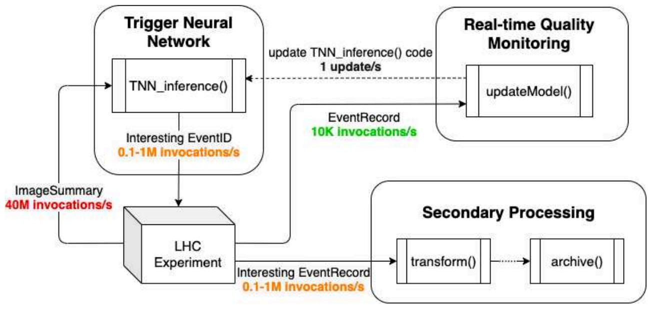


Figure 2: Serverless Software Architecture for LHCTriggering

Comparing Figure 1 and Figure 2, the Trigger Neural Network is mapped to function `TNN_inference()`, which receives an image summary from LHC Experiment as input, performs classification, and responds with an event ID if it contains an interesting signal. The Secondary Processing pipeline maps to a set of functions that process event data in a chained invocation manner. Real-time Quality Monitoring is implemented by `updateModel()`, which is invoked with sampled event records and if necessary, updates function `TNN_inference()`. Depending on the actual algorithm, it may be implemented in multiple functions, and involve the use of external storage to maintain its state.

2.4 Performance Requirements

Such scientific data streaming tasks demand a high frequency of processing on large bandwidth of event streams. All events must be processed within a hard real-time deadline. For the serverless architecture, it implies the following performance requirements:

- *High invocation rate.* To avoid an infinitely growing request queue, functions need to be invoked at the input rate.
- *Low latency.* Functions are subject to hard deadlines, and the tolerated latency is low.

Such extreme requirement is a tough, even impossible, challenge for current serverless implementations. For example, the Trigger Neural Network in LHC event filtering requires 40 million inferences per second, each to be finished within 300 nanoseconds. Three FPGAs are installed at the LHC, each with 6,840 DSPs clocked at 200 MHz [44], which can do 20,520 integer multiply-add operations in one cycle (5 ns), or 1200K operations in 300 ns, that supports the pipelined neural network implementation.

A high-end server CPU such as Intel's Ice Lake SP (40 cores) is capable of approximately 2,944 Int16 Gigaops/second, giving it a capacity of approximately 900K operations in 300ns or 9000K operations in 3 microseconds. This is already competitive with the FPGA in throughput. An NVIDIA A100 GPU [30] is quoted at 624 teraflops of fp/int16 performance. In a 300ns window, it can achieve 180 million operations in 300ns or 1.8 billion operations in 3 microseconds. This shows that the GPU also has sufficient throughput to meet the needs of these streaming applications.

So while these performance requirements seem fantastical, in fact even a single chip of these conventional platforms has more than enough throughput to achieve these rates. However, there is the problem of latency, achieving the 300ns latency is probably not realistic due to current serverless software overheads, batching, and other hardware computation structure requirements needed to achieve high performance. The upgrade to the LHC sensors in the next generation LHC detector provides increased buffering, and hence a looser deadline of 3 microseconds [10]. On the other hand, it is always possible to encapsulate high performance specialized hardware as functions in serverless runtimes. To achieve this, we need radically more efficient software architectures and implementation for serverless, such as custom runtimes and dedicated containers [37].

2.5 Workload Model

Scientific data filtering and analysis workloads can be modelled for serverless systems with the following parameters:

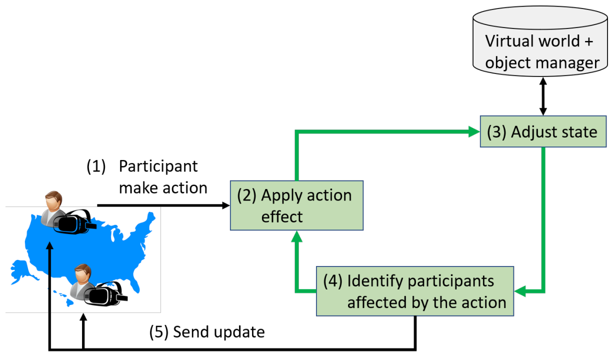

Virtual world + object manager
(3) Adjust state
(1) Participant make action
(2) Apply action effect
(4) Identify participants affected by the action
(5) Send update

Figure 3: Distributed VR/AR Architecture inspired by [8, 16]

- N_t - Event rate, 30 Hz to 40 Mhz, this corresponds to the invocation rate for `TNN_inference()` in the LHC event filtering workload
- Op Associated computation for each event, 5K to 1000K operations
- S_I - Size of an event summary, 2KB to 1MB
- L_t - Latency limit for for filtering decision, currently 300ns
- N_s - Sampling rate for quality monitoring, 1-100 kHz, correponds to the invocation rate for `updateModel()` in the LHC event filtering workload

With these parameters, one can build a workload generator to study serverless system performance with workloads demonstrating these characteristics. We call the serverless implementation meeting the performance requirements when all invocation rates and hard latency limit are met.

3 DISTRIBUTED VIRTUAL REALITY/AUGMENTED REALITY

Distributed Virtual Reality (VR) / Augmented Reality (AR) applications allow many geographically distributed participants to join and interact in a virtual world, either artificial (VR) or blending over the real-world (AR), through a set of actions (e.g. speaking, touching, moving, etc.). Such actions are made independently and simultaneously by individual participants and all together affect the virtual world structure. VR/AR applications capture these actions, apply their effects, and inform participants that should experience the changes by continuously re-rendering their surrounding 3D images. Participants who receive the changes can react with another action, essentially creating a loop of action-update-action interaction. As most VR/AR applications are highly interactive, smoothly handling the action-update process is crucial for good user experiences.

3.1 Application Descriptions

Distributed VR/AR operations mostly surround *Virtual worlds* which are 3D spaces that contain many 3D objects. These objects either represent real-object or are totally made up by the application. *Users* or *Participants* are placed in this world as virtual objects and can interact with other objects through making *actions* such as moving, touching, speaking, etc. User actions trigger the application to *update* the virtual world to reflect the action effect. Figure 3 depicts a typical distributed VR/AR architecture showing how interactions are handled in 5 steps:

(1) Participants make an action at their end device, asynchronously initiating an action request.
(2) Action request are processed by an action handler to resolve its effect.
(3) The application synchronizes actions (if needed) then adjusts the virtual world to reflect action effects.
(4) The application identifies participants who should experience the update.
(5) Updates are sent to proper participants and the application goes back to resolve new actions, starting a new action-update circle.

Since VR/AR are highly interactive, participant actions may become very intensive as we will see from the examples below.

3.2 Examples

Pokemon GO [29] is a great example of massive distributed AR applications. It is a mobile multi-player AR-based game that players can hold their phones, walk around the real world to find and catch virtual characters, call Pokemons, rendered by the game through the phone's camera view. Pokemons are located at some specific geographic locations and only appear to players when they walk around these areas. Since Pokemon GO virtual world is constructed upon the real world, it is vast enough to let millions of players/participants join and interact. They make actions by simply walking and looking for new Pokemon through the phone camera. Pokemon app will keep track of player's movement and response with the appearance of Pokemon or collectible items as rewards.

Despite simple gameplay, the game was a big hit right after it was released with more than 10 million downloads after the first week. The game is still popular nowadays. By 2020, Pokemon GO has approximately 600 million active players worldwide [40]. Even with a simple action-update design, this number is very highly demanding: suppose an average person walks around 7,000 steps a day and Pokemon GO app generates an action for every 10 of them; if only 1% of active players intensively play the game daily then there are up to 4.2 billion actions can be made a day, an average rate of 50,000 requests per second! This demand is not uniform, however. According to [40], Pacific Asia accounts for 52% of global players so this region is far more demanding than Central and Latin America, where has only 9%. Furthermore, in 2019, Pokemon GO held 77 events, and one of them, Safari Zone New Taipei City, attracted 327,000 attendees with 50 million Pokemon caught, potentially generating a burst of thousands of actions per second in Taiwan alone. And in the most extreme case, soon after its release in 2016, Pokemon received a never-seen-before demand increase of 50x their expected load, causing severe experience disruptions for days before it successfully upgraded in Google Cloud and became one of their biggest services since then [7].

Stand Out: VR Battle Royale [33] is another example. It is a VR first-person shooter battle royale that can allow up to 40 players/participants to combat over a wide battlefield that is totally made up by the game designers. Unlike Pokemon GO, the game lets participants make actions in a much more intensive way: to win, they have to act (run, shoot, hide, etc.) fast to take others down. If participants make actions by pressing their controller keys then it is possible to have up to multiple key presses per sec during combat resulting 100 or higher action per second per match. If there are 500 matches are hosted concurrently, then the total rate can go up to 50,000 actions per second, comparable with Pokemon GO load worldwide with merely 20,000 participants! As VR/AR applications are becoming more popular, we are expecting to see a more aggressive load increase in the future.

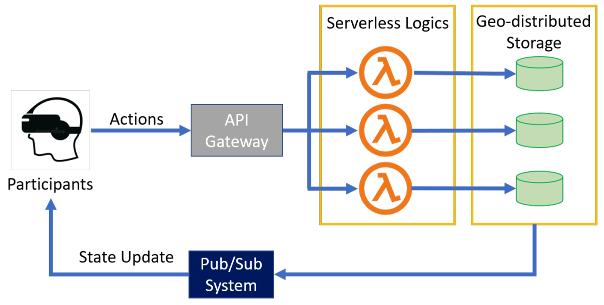


Figure 4: Mapping Distributed VR/AR to Serverless Architecture

3.3 Mapping to Serverless Architecture

Hosting VR/AR application components over serverless is attractive, at least from the cost perspective as it allows the applications to scale the cost to support their highly varied demand down to actual use. Doing so is straightforward, as shown in Figure 3. Each action is mapped to one serverless invocation. After a participant makes an action via their VR/AR end device, the action is sent to an API Gateway. This component collects actions and redirects them to functions located at appropriate locations. Serverless invocations are invoked upon action arrivals to apply their effect, synchronize with the virtual world structure persisted on a geo-distributed storage. Update on the virtual world will trigger a pub/sub system that continuously listens to changes inside the virtual world to inform affected participants.

3.4 Performance Requirements

Since most VR/AR applications are interactive, maintaining a good interactive experience is crucial. From the performance perspective, this is about smoothly responding to/updating participant actions, and can be done by specifying certain requirements around the following metrics:

- *High update/rendering rate.* Participant experience becomes smoother when his view of the virtual world refreshes more often. Low update rate, especially variable, damages effective tracking of changes, reducing the quality of experience. For high quality, an update/rendering rate of 30 to 60 fps is required.
- *Low latency.* Each update is perceptible to the participant. Low latency enables rendering at high rate without skipping or missing changes, improving the interaction quality. To meet the update rate requirement of 30 to 60 fps, the latency requirement should be around 15-30ms. In some cases, this

can be a statistical requirement (e.g. 1% of the frames could be late), or qualitative requirements on the "shape" of the distribution.

- *Scalability*: participant experience must be robust, as the system scales to large numbers of participants, and distributions of those participants in both the interaction space and geographic space.

Given the architecture mapping and requirements above, implementing Pokemon GO with serverless will generate a load of 50,000 invocations per second, each need to be done within 15-30ms. For Stand Out, the demand can be much more if their active users increases. However, with 100+ milliseconds of invocation latency that the current commercial serverless implementation are experiencing nowadays, meeting this latency requirement is challenging [42]. It is even harder as action arrivals are so intensive bursty in both temporal and spatial that requires careful resource allocation and load distribution design.

3.5 Distributed AR/VR Workload Model

VR/AR workloads can be modeled as a sequence of actions initiated by participants. The work depends on the type of actions, and the clustering of other participants in the interaction space. This can be characterized by the following parameters.

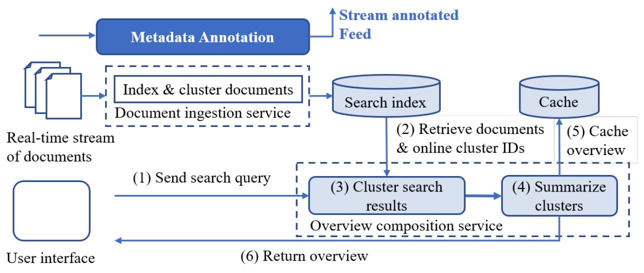
- λ_i – Action arrival rate for participant i, reflecting the frequency of making actions (and serverless invocation). The arrivals should be bursty. We model this as a Poisson process with the average rate raging from 0.01 to 10 per second, depending on participants' aggressiveness of making actions.
- L – Location distribution of participants in the virtual world. L can be formed as a set of multiple clusters of participants (e.g. region, see example above), each is modeled as a two-variable Gaussian distribution with standard deviation vary from 0.1x to 10x of the mean depending on the cluster density.
- $P(L)$ – The distribution of the number of participant affected by a given action, and depends on L, the location distribution
- $F(L)$ – The work embodied in the serverless functions that are required to respond to an action. It is also a function of the participant location distribution L, capturing the fact that more affected participants required more effort on synchronization and update.
- D – Update deadline, which is the maximum latency for any action. To meet the 30-60 fps requirement, we set the deadline within 15-30ms.

We define the following metrics to evaluate application performance over a given workload:

- M – Miss rate, the fraction of actions that take longer than the deadline D. Typical miss rate targets are 5% and 1% or lower.
- $Used$ – Quantity of resource used, for serverless it is in GB*second, to support the workload

4 DOCUMENT ANNOTATION

To support intelligent human decision making, numerous organizations have built text document annotation systems that pore through enormous numbers of documents, organizing them with annotations [6, 24, 25], and providing a large range of derivative summaries. These systems are under performance pressure because of the rapid rate of information growth in the world, and the need to make fast decisions in a competitive landscape, i.e. financial trading. A document annotation application processes millions of documents everyday.



Metadata Annotation
Stream annotated Feed
Index & cluster documents
Document ingestion service
Search index
Cache
Real-time stream of documents
(2) Retrieve documents & online cluster IDs
(5) Cache overview
(1) Send search query
(3) Cluster search results
(4) Summarize clusters
Overview composition service
User interface
(6) Return overview

Figure 5: Text Document Annotation Pipeline from [5] that shows how annotated documents are deposited in a database. They are also streamed to customers in real-time (see blue arrows).

4.1 Application Description

We describe the Bloomberg Financial text annotation pipeline which processes 2 million documents a day. Bloomberg is a real-time information service, and to a large degree, they are annotating documents that are publicly released (company earnings reports, filings, government/fed/treasury statements, business press, and more). In order to provide value to their customers for their very expensive Bloomberg terminals, their document annotation system has a hard deadline of 100ms and no lower bound on desired lower latency (see Figure 5). Examples of processing include natural language processing, specific features (e.g. earnings), topic formation and classification, summarization, and more. Some annotations are done in parallel to minimize the latency, and others may follow a workflow that depends on its content – topics and type of document. The heavy use of natural language means that growing deep learning models such as BERT and GPT-3 are often used. The Bloomberg annotation workflow has a hard deadline of 100ms – though no latency is too low! The objective is to meet ALL of the deadlines with the lowest resource cost.

Each document input is a short collection of text written in natural language, typically 50 to 750 words. In the higher-end examples, images can be included. Annotation is application output that contains information extracted from the document and may relate it to other information in the knowledge base. Workflow is a set of tasks performed by the application to generate annotations for a document. Tasks may depend on each other. Document workflows may interact with each other. A number of papers have been written about a variety of real-time annotations [5, 43]. While each application differs, the basic structure and properties are representative of all of the applications mentioned below.

Annotating a document uses NLP techniques combined with ML/DL models. A standard pipeline is

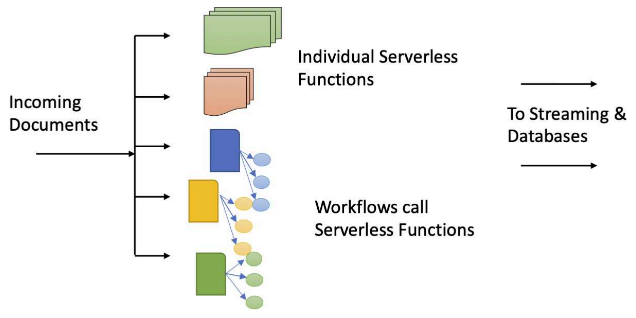

Incoming Documents
Individual Serverless Functions
To Streaming & Databases
Workflows call Serverless Functions

Figure 6: The text annotation application cast into a serverless architecture has key parameters for each function and workflow, as well as their number. Another key parameter is the deadline.

(1) Pre-processing, including parsing [36, 43], cleaning, and tokenization [23], etc. Tasks dependency in the step is predefined (e.g. Parsing $\longrightarrow$ Cleaning $\longrightarrow$ Tokenization). Depending on the annotation purposes, some tasks may be skipped by the workflow.
(2) Feature extraction, with tasks vary depending on the purpose of annotation, and models used in downstream steps. Generally, this step contains POS tagging [24], word embedding construction [24, 26], etc. Tasks are independent. Depending on the annotation purpose, some tasks may be skipped by the workflow.
(3) Annotating the document from selected features using pretrained ML/DL models for natural language [5], topic analysis [34], sentiment analysis [32] and more. Tasks are generally independent, and some are large. For example, state of the art models such as BERT or GPT-3 for NLP requires 200 ms with batch size 8 on an AWS GPU instance [21].[1]

4.2 Examples

While document annotation for financial information services may aggressively optimize to create investment or trading advantage, there are several other at-scale high-performance document annotation problems. For example, Facebook analyzes posts in real-time filtering for a variety of properties (hate speech, interest, ...), and a more ambitious non-real-time annotation for a variety of other properties such as targeted advertisement, user classification, and so on. In another setting, Google and other search engines take real-time feeds from Twitter, news services, new website pages, and more, performing similar real-time document annotation and incremental indexing to provide the ability to search the latest postings and content. We summarize these examples below.

(1) Bloomberg Financial Text Document Annotation (real-time): 2 million documents/day, multiple sources, geographically distributed, real-time, <100ms, no lower bound on latency.
(2) Facebook Post Annotation (real-time): 350 million posts/day, 4,000 per second, geographically distributed, real-time and near real-time, no lower bound on latency [38].
(3) Tweet/article/Web Page Annotation (high rate, near real time): 500 million Tweets/day, 6,000/second, 2 million news articles/day, 24/second, 10 million new pages added to average of 55 billion pages, near-real-time to support incremental search index update for fast-moving content [31, 39].

4.3 Mapping into Serverless Architecture

The document annotation pipeline maps naturally onto an event-driven serverless architecture. When each document arrives, a set of individual functions and workflows are launched. These functions vary in size, and the workflows are a sequence of functions. This mapping is illustrated in Figure 6. The documents vary in size depending on application domain and may also include images as in Facebook posts. Processing times for functions range from short, real-time to longer analysis. Some applications partition their processing into short real-time functions (e.g. hate speech detection) and longer running analysis (e.g. interest classification for targeting).

4.4 Performance Requirements

The application/system is subject to hard real-time deadlines. Its cost is determined by how much computation resources and types of resources (best effort, real-time, etc.) are required to meet its constraints. All documents must meet their deadlines, or the system FAILs. Performance requirements of the system include:

- Invocation rate
- Latency to completed annotation (hard deadline, or 99th percentile)
- Cost: dedicated resources

4.5 Document Annotation Workload Model

Text annotations can be modeled as a series of document arrival, the simplest model would be memoryless (Poisson), but more realistic models would be bursty. The document processing time is typically proportional to size, with modest variation.

- λ - Document arrival rate – memoryless (poisson) process or one with more bursty, correlated arrivals
- S - Document size distribution, ranging from 40 bytes to 4 megabytes (including images). Uniform distributions, or skewed depending on the application.
- $< F_1, F_2, ...F_n >$ - a collection of serverless functions that are invoked on each document. These serverless functions have a corresponding runtime which is proportional to the document size, runtimes distributed across 0.1D to 09D.
- $< W_1, W_2, ...W_n >$ - a collection of workflows each consisting of a set of serverless functions. Again the serverless functions with runtimes proportional to the document size. Runtimes distributed across 0.5D to 0.95D.
- D – hard deadline of 100 milliseconds for text document annotation, soft 99th percentile deadlines of a few seconds for other applications.

Performance metrics for a text document annotation workload include:

[1]Interestingly, also estimated at 1000 request/$.

Workload	Invocation Rate (per second)	Latency Requirement	Scalability
Scientific Stream	40M	300ns (hard)	N.A.
Distributed VR/AR	50K	15ms (soft)	Users
Doc Annotation	12K	200ms (hard)	Annotation Complexity

Table 1: Comparison of Performance Requirements

- F_l – the fraction of documents who miss the deadline; failing to complete annotation within the specified D period after arrival
- R – quantity of resources allocated to support the document annotation to meet deadlines

5 RELATED WORK

Many efforts have been spent on applying the serverless model for a wide range of applications, including video processing [15], data analytics [27], modifiable virtual environments [11] and high-performance computing [9, 37]. These applications, as well as the ones presented in this paper, are the motivation for many studies on serverless that optimize the framework towards high performance [1, 12, 18, 22, 35], enable QoS [41], understand its potentials and limitations [19], as well as to characterize implementations [42] and workloads [35].

Meanwhile, there are studies focusing on serverless application quality in terms of performance guarantee. Batch [2] uses an optimizer to provide tail latency guarantees for machine learning inference. Real-time Serverless [28] provides invocation rate guarantee for bursty, real-time workloads. Spock [17] exploits VMs and serverless functions at the same time to meet SLO at low cost. These are critical steps towards performance predictability, which is also a core requirement of the workloads we presented in this paper.

6 SUMMARY AND FUTURE WORK

Table 1 summarizes the three classes of applications we discussed. They all have real-time (either soft or hard) sub-second latency requirements and scale from tens of thousands to millions of invocations per second. Such demands are far higher than typical serverless workloads, which barely reach 1 invocation per second [35]. Although it is natural to architect these applications on serverless platforms, the performance requirements are far in excess of today's commercial implementations. For example, latency requirements of a few milliseconds versus commercial implementations that are still struggling at 100's of milliseconds to seconds overhead with high variability [42]. The current best-effort allocation employed by serverless platforms also finds it hard to reach or maintain these high invocation rates [28].

Therefore, to support these high demanding applications, a high-performance serverless implementation is needed. We leave find such implementation as an open question as there are many promising directions worth exploring: reducing function initialization overhead, employing load prediction to make proper pre-allocation, adding support for heterogeneous resources such as hardware accelerators, or bringing serverless instances closer to the caller (edge deployment), etc. The serverless platform also needs to provide applications ways to engineer their performance guarantee. This requires better resource isolation to eliminate variability and allocation guarantees to maintain performance requirements under uncertainties.

ACKNOWLEDGMENTS

This work supported by National Science Foundation Grants CMMI-1832230, CNS-1901466, and CCF-1909364. We gratefully acknowledge support from Intel, Google, Samsung, and the CERES Center for Unstoppable Computing.

REFERENCES

[1] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. 2018. SAND: Towards High-Performance Serverless Computing. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 923–935.

[2] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. 2020. Batch: Machine Learning Inference Serving on Serverless Platforms with Adaptive Batching. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, Georgia) *(SC '20)*. IEEE Press, Article 69, 15 pages.

[3] Lixiang Ao, Liz Izhikevich, Geoffrey M. Voelker, and George Porter. 2018. Sprocket: A Serverless Video Processing Framework. In *Proceedings of the ACM Symposium on Cloud Computing* (Carlsbad, CA, USA) *(SoCC '18)*. Association for Computing Machinery, New York, NY, USA, 263–274.

[4] G. Apollinari, I. Béjar Alonso, O. Brüning, P. Fessia, M. Lamont, L. Rossi, and L. Tavian (Eds.). 2017. High-Luminosity Large Hadron Collider (HL-LHC): Technical Design Report V. 0.1. 4/2017 (2017).

[5] Joshua Bambrick, Minjie Xu, Andy Almonte, Igor Malioutov, Guim Perarnau, Vittorio Selo, and Iat Chong Chan. 2020. NSTM: Real-Time Query-Driven News Overview Composition at Bloomberg. *arXiv preprint arXiv:2006.01117* (2020).

[6] Joshua Bambrick, Minjie Xu, Andy Almonte, Igor Malioutov, Guim Perarnau, Vittorio Selo, and Iat Chong Chan. 2020. NSTM: Real-Time Query-Driven News Overview Composition at Bloomberg. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Association for Computational Linguistics, Online, 350–361.

[7] Betsy Beyer, Niall Richard Murphy, David K Rensin, Kent Kawahara, and Stephen Thorne. 2018. *The site reliability workbook: practical ways to implement SRE*. " O'Reilly Media, Inc.".

[8] Chih-Yao Chang, Bo-I Chuang, Chi-Chun Hsia, Wen-Cheng Chen, and Min-Chun Hu. 2020. Framework Design for Multiplayer Motion Sensing Game in Mixture Reality. In *International Conference on Multimedia Modeling*. Springer, 703–708.

[9] Ryan Chard, Yadu Babuji, Zhuozhao Li, Tyler Skluzacek, Anna Woodard, Ben Blaiszik, Ian Foster, and Kyle Chard. 2020. FuncX: A Federated Function Serving Fabric for Science. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing* (Stockholm, Sweden) *(HPDC '20)*. Association for Computing Machinery, New York, NY, USA, 65–76.

[10] Nhan V Tran David W. Miller. [n.d.]. Personal communication.

[11] Jesse Donkervliet, Animesh Trivedi, and Alexandru Iosup. 2020. Towards Supporting Millions of Users in Modifiable Virtual Environments by Redesigning Minecraft-Like Games as Serverless Systems. In *Proceedings of the 12th USENIX Conference on Hot Topics in Cloud Computing*.

[12] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. 2020. Catalyzer: Sub-Millisecond Startup for Serverless Computing with Initialization-Less Booting. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 467–481.

[13] Lyndon Evans and Philip Bryant. 2008. LHC Machine. *Journal of Instrumentation* 3, 08 (aug 2008), S08001–S08001.

[14] Sadjad Fouladi, Francisco Romero, Dan Iter, Qian Li, Shuvo Chatterjee, Christos Kozyrakis, Matei Zaharia, and Keith Winstein. 2019. From Laptop to Lambda: Outsourcing Everyday Jobs to Thousands of Transient Functional Containers.

In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA, 475–488.

[15] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. 2017. Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 363–376.

[16] Frank Glinka, Alexander Ploss, Sergei Gorlatch, and Jens Müller-Iden. 2008. High-level development of multiserver online games. *International Journal of Computer Games Technology* 2008 (2008).

[17] J. R. Gunasekaran, P. Thinakaran, M. T. Kandemir, B. Urgaonkar, G. Kesidis, and C. Das. 2019. Spock: Exploiting Serverless Functions for SLO and Cost Aware Resource Procurement in Public Cloud. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. 199–208.

[18] Zhipeng Jia and Emmett Witchel. 2021. *Nightcore: Efficient and Scalable Serverless Computing for Latency-Sensitive, Interactive Microservices*. Association for Computing Machinery, New York, NY, USA, 152–166.

[19] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Menezes Carreira, Karl Krauth, Neeraja Yadwadkar, Joseph Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. 2019. *Cloud Programming Simplified: A Berkeley View on Serverless Computing*. Technical Report UCB/EECS-2019-3. EECS Department, University of California, Berkeley.

[20] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. 2018. Pocket: Elastic Ephemeral Storage for Serverless Analytics. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 427–444.

[21] Chuan Li. [n.d.]. OpenAI's GPT-3 Language Model: A Technical Overview. lambdalabs.com.

[22] Junfeng Li, Sameer G. Kulkarni, K. K. Ramakrishnan, and Dan Li. 2019. Understanding Open Source Serverless Platforms: Design Considerations and Performance. In *Proceedings of the 5th International Workshop on Serverless Computing* (Davis, CA, USA) *(WOSC '19)*. Association for Computing Machinery, New York, NY, USA, 37–42.

[23] Mounica Maddela, Wei Xu, and Daniel Preoţiuc-Pietro. 2019. Multi-task Pairwise Neural Ranking for Hashtag Segmentation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 2538–2549.

[24] Debanjan Mahata, John Kuriakose, Rajiv Ratn Shah, and Roger Zimmermann. 2018. Key2Vec: Automatic Ranked Keyphrase Extraction from Scientific Articles using Phrase Embeddings. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, 634–639.

[25] Philipp Meerkamp and Zhengyi Zhou. 2016. Information Extraction with Character-level Neural Networks and Free Noisy Supervision. *arXiv preprint arXiv:1612.04118* (2016).

[26] Yishu Miao, Lei Yu, and Phil Blunsom. 2016. Neural variational inference for text processing. In *International conference on machine learning*. PMLR, 1727–1736.

[27] Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 115–130.

[28] Hai Duc Nguyen, Chaojie Zhang, Zhujun Xiao, and Andrew A. Chien. 2019. Real-Time Serverless: Enabling Application Performance Guarantees. In *Proceedings of the 5th International Workshop on Serverless Computing* (Davis, CA, USA) *(WOSC '19)*. Association for Computing Machinery, New York, NY, USA, 1–6.

[29] Niantic. [n.d.]. Pokemon GO. https://pokemongolive.com.

[30] NVIDIA. 2021. NVIDIA A100 Datasheet. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet.pdf.

[31] Daniel Peng and Frank Dabek. 2010. Large-Scale Incremental Processing Using Distributed Transactions and Notifications. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation* (Vancouver, BC, Canada) *(OSDI'10)*. USENIX Association, USA, 251–264.

[32] Daniel Preoţiuc-Pietro, Ye Liu, Daniel Hopkins, and Lyle Ungar. 2017. Beyond binary labels: political ideology prediction of twitter users. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 729–740.

[33] raptor lab. [n.d.]. Stand Out: VR Battle Royale. https://store.steampowered.com/app/748370/STAND_OUT__VR_Battle_Royale/.

[34] Antonia Saravanou, Giorgio Stefanoni, and Edgar Meij. 2020. Identifying Notable News Stories. In *European Conference on Information Retrieval*. Springer, 352–358.

[35] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 205–218.

[36] Tianze Shi, Igor Malioutov, and Ozan Irsoy. 2020. Semantic Role Labeling as Syntactic Dependency Parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, 7551–7571.

[37] Tyler J. Skluzacek, Ryan Chard, Ryan Wong, Zhuozhao Li, Yadu N. Babuji, Logan Ward, Ben Blaiszik, Kyle Chard, and Ian Foster. 2019. Serverless Workflows for Indexing Large Scientific Data. In *Proceedings of the 5th International Workshop on Serverless Computing* (Davis, CA, USA) *(WOSC '19)*. Association for Computing Machinery, New York, NY, USA, 43–48.

[38] Kit Smith. [n.d.]. 60 Incredible and Interesting Twitter Stats and Statistics. https://www.brandwatch.com/blog/facebook-statistics/.

[39] Kit Smith. [n.d.]. 60 Incredible and Interesting Twitter Stats and Statistics. https://www.brandwatch.com/blog/twitter-stats-and-statistics/.

[40] Statista. [n.d.]. Number of active users of Pokémon Go worldwide from 2016 to 2020, by region. https://www.statista.com/statistics/665640/pokemon-go-global-android-apple-users/.

[41] Ali Tariq, Austin Pahl, Sharat Nimmagadda, Eric Rozner, and Siddharth Lanka. 2020. Sequoia: Enabling Quality-of-Service in Serverless Computing. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (Virtual Event, USA) *(SoCC '20)*. Association for Computing Machinery, New York, NY, USA, 311–327.

[42] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking Behind the Curtains of Serverless Platforms. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 133–146.

[43] Chunyang Xiao, Christoph Teichmann, and Konstantine Arkoudas. 2019. Grammatical Sequence Prediction for Real-Time Neural Semantic Parsing. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*. 14–23.

[44] Xilinx. [n.d.]. UltraScale+ FPGA Product Tables and Product Selection Guide. https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf.